On Transaction Design for UML Components

Sten Loecher
Dresden University of Technology
Sten.Loecher@inf.tu-dresden.de

Abstract: The transaction concept enables the efficient development of concurrent and fault tolerant applications. Transaction services are therefore an essential part of modern component technologies, such as Enterprise JavaBeans, which are used to develop server-side business applications. The container, which is the execution environment of component-based applications, provides the services and uses corresponding configuration information to apply them properly. The required transactional behavior can be specified by using pre-defined configuration attributes. A result of this declarative approach are separate workflows for designing business and transaction logic. In this paper, we argue that existing methods for component-based development lack adequate support for transaction design. We then describe a model-driven process, which is directed towards an integration with the UML components method elaborated by Cheesman and Daniels.

1 Introduction

Basic requirements to be fulfilled by business applications, such as online shopping systems or electronic banking applications, are efficient and reliable execution of user requests. From a technical point of view, this implies concurrent execution of processes and employment of fault tolerance mechanisms. Transactions address these issues by providing the so-called ACID properties to an application, which stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability. In other words, they guarantee atomic and isolated execution of processes, facilitate consistent system states after process execution, and durability of process results after commitment[GR93].

Since transactions simplify the development of concurrent and fault tolerant systems significantly, they are an essential part in modern component technologies, such as Enterprise JavaBeans (EJB)[DmYK01], which provide the technological foundation for efficiently developing server-side business applications. A basic characteristic of these technologies is the differentiation between the actual business application and the runtime environment, the so-called container, which provides infrastructure services like transaction management. This architecture enables separation of concerns with regard to application-specific and application-independent logic in the development process. The integration of both aspects can be performed by either inserting control statements into the application-specific code or by declaratively configuring the application using pre-defined configuration attributes.

In our work, we *focus on the declarative configuration of transaction management services*. We consider this approach more suitable for an efficient software development process than the programmatic approach. This is due to the better separation of concerns, which enables independent processing of the business specific and the transaction specific aspect by domain experts and configuration of transaction management services based on simple, yet precise, configuration mechanisms.

Efficient development of software systems does not only depend on the technological foundation. An accompanying development method is a crucial factor for successfully applying a technology. For the case of component technology, a corresponding method must include both a process for developing component-based applications and a process for handling the separated transactional aspect and its integration. However, current methods either focus on component-based development[DW98, Se03, CD00] or aspect orientation[Ki96, PC01].

For example, the development method proposed by Cheesman and Daniels in [CD00], to which we refer to as UML components method in this paper, defines a process for developing and specifying component-based business applications. It provides a concise description of the workflows and related activities as well as concrete advice for performing the individual tasks in the development process. However, design of transactional logic is not considered sufficiently. It is just informally stated how transactions could be designed and it is supposed that the actual configuration process takes place not until the assembling of the application.

We think that the transaction design process and its integration with the design process for component-based applications must be defined more precisely. This is required for transactions to exploit their full capabilities with regard to concurrency and fault tolerance, which implies conscientious design. Therefore, we define a process for transaction design in this paper. The result of transaction design are specifications describing transaction demarcation, dependencies between transactions, and the behavior in case of failure. The *process is model-driven* and therefore allows an early recognition of inappropriate transaction design properties and design errors based on model analysis. The integration with component-based development is discussed using the UML components method by Cheesman and Daniels.

The presented process description is part of our work on elaborating the foundations for model-driven transaction design[LH03, Lo04b, Lo04a]. We contribute to the definition of a complete development process for component-based server-side business applications and show how to integrate aspect-oriented and component-based development for the case of transaction management services. Furthermore, the described process contributes to the discussion about Model-Driven Architecture MDA[MM03] with respect to model-driven configuration of infrastructure services, which is a prerequisite for a complete model-driven development process.

After a discussion of some preliminary issues in the following section, the process for transaction design is introduced in Sect. 3. The presented work is related to existing literature in Sect. 4. We conclude in Sect. 5 with a summary of the paper.

2 Preliminaries

For a better understanding of Sect. 3, a number of preliminary issues are explained. First, we explicitly state our assumptions about configuration mechanisms and container capabilities with regard to transaction services. Since our process description is based on terms defined by the Rational Unified Process (RUP)[Kr98], we introduce the essential basics in Sect. 2.2. An overview of the UML components approach is provided in Sect. 2.3. Finally, an example application is introduced in Section 2.4, which is used to illustrate the transaction design process.

2.1 Configuration Mechanisms and Container Services

The transaction design process presented in this paper is elaborated from a software engineering perspective, i.e., with the goal to provide efficient process support for software engineers that develop application software. Therefore, we make two assumptions. First, we assume that *various specific types of transaction configuration models* are available for software engineers. Second, we require the *container to support the nested or advanced transaction models.*

In [LH03] and [Lo04a] we have discussed the requirement to provide different tailored transaction configuration mechanisms to software engineers. These mechanism are captured by different types of configuration models that support different viewpoints on the system and the use of various transaction models during configuration. Different viewpoints on the system to be developed result from the different workers that are involved in the transaction design process and their required level of abstraction, as well as their scope of knowledge with regard to single components, groups of components, or the whole system. Section 3 provides examples of configuration models that we have developed in our work.

Support for various transaction models is required to build efficiently executable applications. Since the flat transaction model has a number of restrictions with regard to functionality and performance, advanced transaction models and techniques[El92] have been developed and applied within database management systems. However, this situation is not reflected in current component technologies like EJB[DmYK01]. We nevertheless assume advanced transactional functionality to be a necessary prerequisite. This point of view is supported by other authors as well[Pr02, SS03] and containers that support advanced transaction models have already been developed[Pr02].

2.2 Rational Unified Process

The Rational Unified Process[Kr98] is a method framework that is used to describe development processes and to assign tasks and responsibilities within a development organization. For this, it defines the development process in terms of activities and workflows that

define the rules for creation or update of artifacts by workers.

Workers define the behavior and responsibilities of individuals or teams for performing tasks during the development process. Typical examples for workers are business designer, architect, and implementer. Note that workers are not individuals but rather roles that can be played by individuals. They are merely used by the project manager to assign tasks to persons. Consequently, persons can be assigned multiple workers. Workers own *artifacts*, which are pieces of information that are produced, modified, or used during the development process, such as design models or actual source code. However, artifacts are not documents. Rather, documents are generated from artifacts.

The actual behavior of workers during the development process is defined by *activities*, which describe the essential tasks to create, use, or update artifacts. An example for an activity is the definition of system interfaces and operations based on provided use cases as explained in [CD00]. Finally, *workflows* are used to define sequences of activities that produce results of observable value and to relate workers to activities. The specification workflow in [CD00], for example, describes the activities necessary to produce component specifications and application architectures, which serve as a basis for component implementation and final application assembly.

2.3 UML Components

The UML components approach to software engineering has been elaborated by Cheesmann and Daniels[CD00]. It is a very simple and practical oriented engineering process for the development and specification of component-based applications. The authors focus on the development of system and business services of N-tier distributed architectures, i.e., the server-side part of distributed business applications. These applications comprise some front end for user interaction, a middle tier providing the system and business services, and backends such as databases for managing application data. *System services* are application specific and designed to implement individual business processes. *Business services* are of a more general nature and thus can be used across multiple applications. In terms of EJB, for example, system services may be implemented by session beans whereas business services may be implemented by entity beans. The functionality is made accessible via accordingly named system and business interfaces.

The development process underlying UML components is aligned with the Rational Unified Process (RUP)[Kr98]. There are basically six workflows in the development process, namely requirements, specification, assembly, provisioning, test, and deployment. Whereas the requirements, test, and deployment workflow relate directly to their counterparts in the RUP, the specification, provisioning, and assembly workflow replace the original analysis, design, and implementation workflows.

In [CD00], the authors focus mainly on the specification workflow, which defines activities to develop component specifications and application architectures from business concept models, use cases, and existing assets with respect to technical constraints. The provisioning of actual components according to specifications and the eventual assembly with

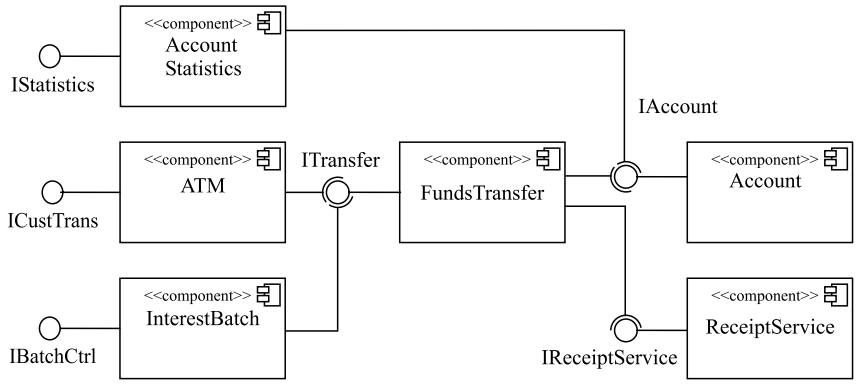

Figure 1: Example application.

respect to the defined application architecture are defined rather sketchily.

The documentation of artifacts within the UML components method is based on Version 1.3 of the UML standard. Since the authors of the UML components method do not make use of extensions and alternative notations as those provided by the standard, we think that an upgrade to the upcoming UML 2.0 standard is not problematic but rather enriches the method.

2.4 Working Example

Throughout the paper, we use an example application to illustrate the transaction design workflows and activities. The application, which is part of a banking system, is illustrated in Figure 1. Its purpose is to serve three use cases, which are the creation of account statistics, the processing of money transfer requests for customers of the bank, and the automated booking of earned interests to bank accounts. Accordingly, the application provides three components that implement the system services, namely `AccountStatistics`, `ATM`, and `InterestBatch`. The latter two components make use of a `FundsTransfer` component that manages the actual money transfer between accounts. Accounts are represented by the `Account` component. The `FundsTransfer` component optionally makes use of a `ReceiptService` component that issues a receipt after successful money transfer processing via a connected printing device. The complete interface definitions are omitted for reasons of brevity but will become obvious in the following sections.

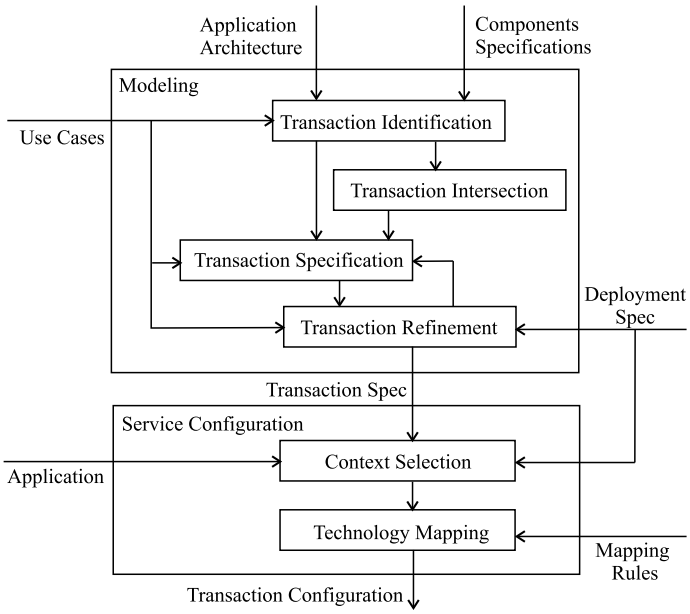

Figure 2: Transaction Design Workflows.

3 Transaction Design

This section introduces the transaction design process. We start in Sect. 3.1 with an overall picture of the process by explaining the workflows and involved artifacts. The integration of the transaction design workflows and the workflows defined by the UML components method is discussed in Sect. 3.2. The individual activities that are performed in the transaction design workflows are explained in Sect. 3.3. Finally Sect. 3.4 defines the required workers and their relations to activities.

3.1 Workflows

The analysis, design, and implementation of transactional logic is captured by two workflows, namely transaction modeling and transaction service configuration. Whereas the transaction modeling workflow realizes the analysis and design of transactional behavior, the purpose of transaction service configuration is to implement a transaction design for a specific environment respectively to configure the environment accordingly. By introducing separate workflows for transaction design, we stress the fact that transaction design is an important aspect of application development that is to be handled by accordingly skilled workers. Figure 2 depicts the workflows and related activities.

3.1.1 Transaction Modeling

Transaction modeling produces transaction specifications that match the required transactional behavior of the application and provides optimization plans for different target environments. For this, the essential activities to be performed are transaction identification, intersection analysis, transaction specification, and transaction refinement. The first three activities result in transaction specifications that match the required transactional behavior of the business application. Transaction refinement eventually elaborates plans to optimize transaction specifications with regard to functional and non-functional requirements for individual environments based on corresponding deployment information.

The required artifacts for transaction modeling are use cases, component specifications, application architectures, and deployment specifications:

- *Use cases* are used during requirements elicitation to capture the interaction structure between the client and the system and to document the tasks to be performed by the system. They are also used to provide information about business transactions required by the client[AM00] and therefore play an important role in determining the overall transactional structure of the system.
- *Component specifications* provide the essential information that is required to analyze and to make decisions about transaction properties of individual components. This includes structural information, such as provided and required interfaces, as well as functional specifications that describe, for example, the causality between provided and required operations and the temporal ordering of operation calls on required interfaces.
- *Application architectures* define software systems in terms of computational components and interactions among those components[SG96]. In other words, they describe networks of interconnected components. Application architectures provide the basis to analyze the propagation of transactions within a system and to reason about resulting properties, such as deadlock freedom.
- *Deployment specifications* are used to define the execution architecture of systems that represent the assignment of components to hardware devices or software execution environments and the connections between such devices and execution environments. This information is required to refine transaction designs with regard to recovery points and inner-transaction parallelism, which is discussed in more detail in Sect. 3.3.

We like to emphasize that we do not require full formal specification of components, architectures and deployment information but specifications sufficiently precise to analyze and to decide about transaction design issues. This is justified by practical experience in software engineering[Hu00]. The transaction modeling workflow results in *transaction specifications* that describe transaction demarcation, dependencies between transactions, and the behavior in case of failures independently of the technical platform.

It is also emphasized that in particular the transaction modeling workflow is not linear but rather an iterative process. The transaction specification and refinement activity, for example, are naturally performed alternately until a usable transaction specification is created.

3.1.2 Transaction Service Configuration

Transaction service configuration is based on transaction specifications and results in actual configurations for specific environments, such as deployment descriptors for EJB containers. Therefore, essential activities are the selection of context specific specifications and the mapping of the corresponding platform independent transaction specification to the target platform. For this, concrete deployment information as well as *mapping rules* are required. The result of the configuration workflow are platform-specific configurations for the application.

3.2 Integration with UML Components

The integration with the workflows of the UML components method is driven by the creation and update of artifacts and the implied chronological order of the workflows.

The *specification workflow* from [CD00] provides the artifacts that are required for *transaction modeling*, which is therefore performed *subsequently* to it. Also, an *alternation* with the specification workflow is possible, because transaction modeling can result in feedback about non-compatible compositions of components with regard to transaction management services. For example, different component realizations can exhibit different properties with regard to possible deadlock of the system and must be chosen accordingly.

The *transaction service configuration workflow* is supposed to be performed *after* the *assembling of the application* and therefore after the corresponding workflow in [CD00], because the configuration is performed on the actual application.

3.3 Activities

This section explains the individual activities that are necessary to perform the transaction modeling and configuration workflow. The activities are introduced in the order they are performed within the workflows. Each activity is first characterized and then explained on the basis of the working example. Besides that, the essential input and output artifacts are stated explicitly.

3.3.1 Transaction Identification

The first step to design the transactional logic of an application is to identify transaction candidates. In other words, atomic processes have to be identified within the system that

is developed. *We follow [Da78], who determines process atomicity by the amount of processing that one wishes to consider as having identity*. For this, [CD00] suggests using operations of system interfaces to identify transaction candidates, i.e., each such operation is considered a potential starting point of a transaction. If a transaction is actually required is determined by execution properties of the operation, such as shared state access. We make use of this idea but refine it to identify distinguished atomic processes within the transaction candidates that require special transaction properties, such as special recovery strategies or vitality properties with respect to the commitment policy of the enclosing transaction. To find such distinguished atomic processes, the call graph of system operations is determined and investigated. To summarize, the transaction identification activity is required to find transaction candidates and is broken into two steps, namely the identification of atomic processes based on use cases and system operations and the identification of distinguished sub processes that require special transaction properties, such as different recovery strategies.
Input: use cases, application architecture, component specifications
Output: transaction candidates
Example: The example system provides three system interfaces. After an analysis of the call dependencies within the business logic, based on the system operations `IStatistics::getAccountsTotal`, `ICustTrans::creditTransfer`, and `IBatchCtrl::processInterest`, the following transaction candidates are identified[1]:

```
<transaction-spec>
  <transaction id=1>
    <root>IStatistics::getAccountsTotal</root>
  </transaction>
  <transaction id=2>
    <root>ICustTrans::creditTransfer</root>
    <demarcation>
      <sub-id>4</sub-id>
      <vitality>non-vital</vitality>
    </demarcation>
  </transaction>
  <transaction id=3>
    <root>IBatchCtrl::processInterest</root>
    <demarcation>
      <sub-id>5</sub-id>
      <vitality>non-vital</vitality>
      <retry>retryable<retry>
    </demarcation>
  </transaction>
  <transaction id=4>
    <root>IReceiptService::printReceipt</root>
    <recovery-type>norecovery</recovery-type>
  </transaction>
```

[1] We use an XML notation to describe the determined atomic processes and their distinguished properties. The underlying metamodel has been defined within our framework for transaction configuration models[Lo04a] and could also be represented by an alternative notation, such as a graphical one.

```
    <transaction id=5>
      <root>ITransfer::transfer</root>
      <recovery-type>backward</recovery-type>
    </transaction>
  </transaction-spec>
```

The model defines five transactions with consecutively numbered identifiers, which we denote T_n (n=1,...,5) in the following. Transactions T_1, T_2, and T_3 result from the system operations, which are declared as root operations of the respective transaction. The used type of model assumes implicit propagation of declared transactions. An analysis of the call graph of operation `ICustTrans::creditTransfer` revealed a distinguished process for printing the money transfer receipt. Since a printed receipt cannot be revoked once printed and therefore has the distinguished property of not being recoverable, the printing operation has been excluded from T_2. For this, T_2 is demarcated at the `printReceipt` operation, which is declared by a reference to T_4 that models the printing transaction.

3.3.2 Transaction Intersection

One reason for using transactions is to enable a system to be used by multiple users concurrently. As a result, transactions can interleave at runtime, i.e., transactions can share components and application data. Transaction intersection information can help software engineers in two ways. First, individual transaction properties can be adjusted based on the analysis results to facilitate efficient concurrent processing of transactions by the system. Isolation levels can be, for example, selected with respect to the required accuracy of transaction results as discussed in [Ew01]. Second, transaction intersection analysis is required to select local configurations properly. This is required, because current component technologies are based on single configuration attributes that are associated to provided operations of components. If operations are executed within different scenarios that require different transactional properties, the configuration must be selected with respect to all scenarios.
Input: transaction candidates
Output: transaction intersection information
Example: By analyzing the possible interleaving between the three transaction candidates, an important point of intersection is identified between T_1 and T_2. Both transactions use the `Account`. However, wheras T_2 is required to run in full isolation since fund transfers must preserve consistency of the involved data, T_1 can be allowed to operate on a lower isolation level and therefore possibly inconsistent data since the result of the calculated statistic is not crucially dependent on individual account data. Also, T_2 and T_3 interleave at the `transfer` operation and the subsequently called operations. For this, the resulting local transaction specification for the `transfer` operation must be a combination of the specifications of T_2 and T_3 that takes the requirements of both transactions into account.

3.3.3 Transaction Specification

The result of transaction identification and transaction intersection analysis provide the starting point for precisely specifying the required transactional behavior of an application. The purpose of the transaction specification activity is to create explicit and precise specifications that describe the required transactional behavior of an application. For this, transaction candidates must be elaborated to complete specifications based on transaction intersection analysis results. Therefore, an important difference between the results of transaction identification and transaction specification regard the incorporation of transaction interaction information. The properties of the resulting transaction specification may vary with respect to the point of view about locality of the software engineer, the available and desired transaction concepts respectively models, and the level of abstraction applied during development[Lo04a].
Input: use cases, transaction candidates, transaction intersection information
Output: transaction specification
Example: To illustrate the transaction specification activity, we use another example of a transaction specification based on XML that allows specification of transactional behavior in terms of provided and required transactional properties of an individual component. Such a specification allows, for example, to be replaced by another specification based on local information exclusively.

```
<transaction-spec>
  <method>
    <method-name>transfer</methode-name>
  </method>
  <configuration>
    <trigger>invocation</trigger>
    <recoverability>backward</recoverability>
    <demarcation>requires</demarcation>
  </configuration>
  <requires>
    <method>
      <method-name>*</methode-name>
    </method>
    <trigger>invocation</trigger>
    <recoverability>backward</recoverability>
    <demarcation>propagate</demarcation>
  </requires>
</transaction-spec>
```

The example specifies that the `transfer` operation of component `FundsTransfer` must be executed within a transaction. If the client of the operation does not call from within a transaction, a transaction must be started upon invocation of `transfer`. Also, the transaction is required to be propagated to the operations called within `transfer`. This specification is derived from the transaction candidates T_2 and T_3. Since T_2 requires the propagation of the transaction context provided by the `ATM` component and T_3 requires the begin of a new transaction upon invocation of `transfer`, the demarcation attribute has been chosen to serve both usage scenarios.

3.3.4 Transaction Refinement

The transaction identification activity determines the system processes for which the ACID properties must be guaranteed at runtime. Transaction specification provides precise and explicit specifications of the corresponding transactional logic. At this point, the transaction modeling workflows could be considered completed. However, for the application to be executed efficiently, it may be required to refine the elaborated specifications with regard to intra-transaction structure and individual transaction properties. In particular, transactions may be structured into subtransactions to increase intra-transaction concurrency and to provide more adequate recovery points. The refinement of transaction specifications is driven by information about the actual deployment context of an application. Such information regards, for example, the distribution of components on different nodes of a network, physical properties of the corresponding hardware, and supported transaction models within the target container. If an application is to be deployed within different target environments, various refinements are possible with respect to the actual deployment context. Basically, there are two strategies to provide multiple tailored transaction specifications. On the one hand, multiple deployment context specific transaction specifications can be created during transaction refinement. The software engineer that configures the application then has to choose the specification that matches the actual deployment context properties most adequately. On the other hand, refinement plans can be provided that are used by the responsible software engineer to derive a deployment context specific configuration. For example, such a plan can provide the rule to generally propagate transactions at individual operations of components or to preferably invoke them within subtransactions if the actual container supports that feature. The actual strategy chosen depends on the specific application or project properties, respectively.
Input: transaction specification, deployment information
Output: transaction specifications, refinement plans
Example: To demonstrate the refinement of transaction specifications, we will have a closer look at transaction T_2 and in particular the subsystem comprising the `FundsTransfer` and `Account` component. The specification elaborated in the preceding activity requires the subsequent execution of the `debit` and `credit` operation, denoted by the wildcard, within one transaction. To increase intra-transaction concurrency and therefore application performance, both operations can be executed within concurrently executed subtransactions. Of course, the component and container must support such behavior. Furthermore, with the knowledge that the communication between the `FundsTransfer` and `Account` components is realized over a communication link with a specific failure rate, a retry policy can be specified, declaring that the debit and credit operation must be recalled several times in case of failure to increases the success rate according to the application requirements. A refined version of the specification for the `transfer` operation may be:

```
<transaction-spec target=1>
  <method>
    <method-name>transfer</methode-name>
  </method>
   ...
```

```
    <requires>
      <method>
        <method-name>*</methode-name>
      </method>
      <trigger>invocation</trigger>
      <recoverability>backward</recoverability>
      <demarcation>sub</demarcation>
      <no-retry>1<no-retry>
    </requires>
  </transaction-spec>
```

The example declares that the specification is meant to be used within a specific target environment, denoted by `target=1`, which is a reference to a corresponding deployment specification that we omit for reasons of brevity. It is specified that the called operations are required to be invoked within subtransactions and should be retried one time in case of failure.

3.3.5 Context Selection

The first activity of the transaction configuration workflow is the selection of a specific transaction specification with regard to the actual deployment context of the application. This selection is either a choice among multiple provided transaction specifications or comprises the selection of a context specific refinement plan and the subsequent refinement of the transaction specification in accordance to the plan.
Input: transaction specification(s), refinement plans, deployment information
Output: deployment context specific transaction specification
Example: Depending on the properties of the target container, either the original specification using only flat transactions or the refined version using nested transactions can be used for deployment. In the example, we choose to deploy the application to an EJB container that supports only flat transactions and therefore select the unrefined specification.

3.3.6 Technology Mapping

In the preceding activities, we have used platform-independent specification models that allow software engineers to efficiently handle the transaction modeling task. By platform independence we mean the abstraction from container-specific configuration mechanisms and notations. That also allows the use of the specifications across multiple platforms, which increases reuse. To use transaction specifications on real platforms, they must be mapped to platform-specific configuration mechanisms. For this mapping, which can be automated by transformation tools, rules must be defined.
Input: transaction specification, mapping rules
Output: technology specific deployment descriptor
Example: To demonstrate the mapping of the selected transaction specification for the `FundsTransfer` component, we choose as target platform a standard conform EJB container. The EJB standard[DmYK01] defines six configuration attributes that can be used

for transaction service configuration. The mapping of the specification to EJB-specific configuration attributes that are stored in XML deployment descriptors has the following results:

- The `ITransfer::transfer()` operation is configured with the EJB attribute `Requires`, since the result of the interpretation by the container is the specified transactional behavior.
- The requirement of executing the required operations of the `Account` component within the same transaction is mapped by configuring the corresponding operations of the `Account` component with the attribute `Mandatory`, since the configuration of required operations of components is not possible in EJB.

The mapping rules for this example are applied intuitively and are not specified explicitly for reasons of brevity. Also, the example is rather simple. However, more complex mappings are possible. For example, more sophisticated container services can be used for transactions, such as developed by Prochazka[Pr02].

3.4 Workers

The transaction modeling and configuration workflows comprise several activities that must be performed by accordingly skilled workers. This section establishes a role model that is used by the project manager to assign tasks to project members.

Generally, the defined activities for transaction modeling and configuration can be categorized into identification, optimization, and configuration. Correspondingly, we introduce three workers that handle the tasks, namely the transaction designer, the transaction optimizer, and the transaction configurator:

Transaction Designer: The transaction designer is responsible for the identification of transactions, intersection analysis, and transaction specification. He must have basic knowledge about the purpose of transactions and must be able to identify key properties of atomic processes. The transaction designer develops an initial transaction specification based on uses cases, component specifications, and application architectures.

Transaction Optimizer: The transaction optimizer is a specially skilled worker who has particular knowledge about transaction optimization. Based on deployment plans and initial transaction specifications he elaborates refined transaction specifications that provide potentially more efficient transactional behavior.

Transaction Configurator: The transaction configurator is an engineer who is supposed to implement a specification based on transaction specifications. Therefore, he is not required to be specially skilled with respect to transaction design. He chooses a transaction specification and maps it, with the help of tools, to the deployment platform.

The workers discussed so far are only those workers directly involved in the transaction modeling and configuration workflow. However, other workers that have responsibilities with regard to transaction design are:

Component Provider: The main task of the component provider is to design and implement components and to deliver them to the application assembler, i.e., he has the responsibility to provide adequate component specifications that enable safe transaction design.

Application Architect: The application architect designs the business application. Also he adapts the application architecture based on technical constraints and feedback from the transaction modeling process. For example, he replaces components so that transactions can be applied more properly.

4 Related Work

The area of component-based development is still evolving. However, there exist several established methods for component-based development, which heavily build on proven concepts from object-oriented modelling methods[SDS01]. Prominent examples are the Catalysis approach[DW98], the Select Perspective method[AF00], and the UML components method[CD00]. Our work is strongly related to the UML components method, which we have chosen as foundation for our work since it provides a very concise description of the development process, is very accessible, and comprehensive.

The situation with aspect-oriented methods is somewhat different. Although aspect-oriented programming (AOP) approaches have been proposed in the literature, their practical applications are still unclear. Thus, there exist no established methods that describe the aspect-oriented development process exhaustively[PC01]. However, several authors work on process descriptions. For example, [PD03] analyzes the effects of integrating aspect-oriented software engineering and the RUP. This work is of interest to us, since we use the RUP as foundation for describing the concepts of the transaction design process.

Finally, our work aims at contributing to the discussion about Model-Driven Architecture [MM03]. There are other projects that also work on model-driven middleware configuration, such as CoSMIC[TG04]. However, they focus on different aspects than transaction management services and often focus on the definition of tool chains to support the development process instead of the development process itself.

5 Conclusion

In this paper, we argue that current methods for component-based development of server-side business applications do not adequately support the design and configuration of transaction management services. We introduce a process for model-driven transaction design that integrates with the UML components method elaborated by Cheesman and Daniels.

The process includes two workflows, namely transaction modeling and transaction configuration. Transaction modeling deals with the design of the transactional logic for an application model. Transaction configuration is concerned with the adequate implementation of a transaction design on a specific target platform. The design space, i.e., available transaction models, recovery strategies, etc., is captured by different types of models used by the developer.

The main contribution of this paper is the description of an integrated development process for component-based applications that use transaction management infrastructure services provided by the runtime environment based on declarative configuration. We also contribute to the elaboration of a complete model-driven development process for component-based development based on the UML components approach. We like to stress that we do not intend to reinvent existing concepts from the database and transaction processing domain, but to bridge the existing gap to software engineering and the application of these concepts within this domain.

So far we have not validated the design process based on complex applications and empirical studies. However, validation of the presented process is supported by our previously presented framework for transaction service configuration[LH03, Lo04b, Lo04a], which provides the technical foundation for a model-driven transaction design process. A first tool prototype to support the approach and to show the applicability of the proposed concepts has already been implemented. We think that, with the position and the results presented in this paper, we provide a proper foundation for a discussion about model-driven service configuration in the context of the Model-Driven Architecture.

References

[AF00] Allen, P. and Frost, S. (eds.): *Component-Based Development for Enterprise Systems, Applying the SELECT Perspective*. Cambridge University Press/SIGS, Cambridge. 2000.

[AM00] Armour, F. and Miller, G.: *Advanced Use Case Modeling: Software Systems*. Addison-Wesley. 2000.

[CD00] Cheesman, J. and Daniels, J.: *UML Components: A Simple Process for Specifying Component-Based Software*. Addison-Wesley. 2000.

[Da78] Davies, Jr., C. T.: Data processing spheres of control. *IBM Systems Journal*. 17(2):179–198. 1978.

[DmYK01] DeMichiel, L. G., mit Yalcinalp, L., and Krishnan, S. (eds.): *Enterprise JavaBeans Specification, Version 2.0*. Sun Microsystems. 2001.

[DW98] D'Souza, D. F. and Wills, A. C.: *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley Object Technology Series. Addison-Wesley Publishing Company. 1998.

[El92] Elmagarmid, A. K. (eds.): *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers. 1992.

[Ew01] Ewald, T.: *Transactional COM+: Building Scalable Applications*. Addison Wesley. February 2001.

[GR93] Gray, J. and Reuter, A.: *Transaction Processing: concepts and techniques*. Morgan Kaufmann Publishers, Inc. 1993.

[Hu00] Hussmann, H.: Towards Practical Support for Component-Based Software Development Using Formal Specification. In: *Modelling Software System Structures in a Fastly Moving Scenario, Workshop Proceedings. Santa Margherita Ligure, Italy*. June 2000.

[Ki96] Kiczales, G.: Aspect-oriented programming. *ACM Comput. Surv.* 28(4es):154. 1996.

[Kr98] Kruchten, P.: *Rational Unified Process: an Introduction*. Addison-Wesley. 1998.

[LH03] Loecher, S. and Hussmann, H.: Metamodelling of Transaction Configurations - Position Paper. In: *Metamodelling for MDA, First International Workshop, York, UK*. University of York. November 2003.

[Lo04a] Loecher, S.: Model-Based Transaction Service Configuration for Component-Based Development. In: *7th Workshop on Component-Based Software Engineering (CBSE7), Edinburgh, Scottland, Workshop Proceedings, to be published in Lecture Notes in Computer Science (LNCS) by Springer.* March 2004.

[Lo04b] Loecher, S.: Modellbasierte Konfiguration von Transaktionsdiensten. In: *Modellierung 2004, Gemeinsame Konferenz von zwoelf Fachgruppen der GI, Marburg, Germany, Proceedings zur Tagung, LNI Volume P-45*. Gesellschaft fuer Informatik. March 2004.

[MM03] Miller, J. and Mukerji, J. (eds.): *MDA Guide Version 1.0*. www.omg.org. May 2003.

[PC01] Pace, J. A. D. and Campo, M. R.: Analyzing the role of aspects in software design. *Commun. ACM*. 44(10):66–73. 2001.

[PD03] Piveta, K. and Devegili, A. J.: Aspects in the rational unified process' analysis and design workflow. 2003.

[Pr02] Prochazka, M.: *Advanced Transactions in Component-Based Software Architectures*. PhD thesis. Charles University, Faculty of Mathematics and Physics, Department of Software Engineering, Prague. 2002.

[SDS01] Stojanovic, Z., Dahanayake, A., and Sol, H.: A Methodology Framework for Component-Based System Development Support. In: *6th CaiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design EMMSAD01, Interlaken, Switzerland*. June 2001.

[Se03] Select Business Solutions. Introducing Select Perspective - Delivering Component Based Solutions. www.selectbs.com. 2003.

[SG96] Shaw, M. and Garlan, D.: *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall. 1996.

[SS03] Silaghi, R. and Strohmeier, A.: Critical Evaluation of the EJB Transaction Model. In: N.Guelfi, E.Astesiano, and G.Reggio (eds.), *Scientific Engineering for Distributed Java Applications. International Workshop, FIDJI 2002 Luxembourg-Kirchberg, Luxembourg, November 28-29, 2002*. volume 2604 of *LNCS*. S. 15–28. Springer. 2003.

[TG04] Turkay, E. and Gokhale, A. Addressing the middleware configuration challenges using model-based techniques. http://www.dre.vanderbilt.edu/cosmic/acmse_mdm_ocml.pdf. April 2004.